Credit Card Fraud Detection

Sebastian Buxman, Harthik Sonpole

December 2024

Problem Description

Credit card fraud detection is a pressing issue in today's society. Fraudulent activities often result in significant monetary losses - the FTC reported $10 billion in losses due to fraud in 2023, which is an increase of $1.2 billion from 2022. Automating fraud detection reduces financial losses and enhances trust in digital payment systems. Our goal was to develop a machine learning model that accurately identifies fraudulent transactions while minimizing false alarms. The model will predict whether a transaction is fraudulent (1) or legitimate (0), making it a binary classification task.

Data Set Description

We chose a dataset called Credit Card Fraud Detection, found on Kaggle. It contains anonymized data from European cardholders and comes from September 2013.

Attributes and Features

- There are 30 predictor variables:
  - 28 anonymized features (V1 to V28), which are derived using Principal Component Analysis (PCA).
  - *Amount:* Numeric, representing the transaction amount.
  - *Time:* Numeric, representing the seconds elapsed since the first transaction.
- There are 284,807 transactions in the dataset:
  - 284,315 (99.83%) instances of non-fraudulent transactions.
  - 492 (0.17%) instances of fraudulent transactions.

```
Training Set Class Distribution:
Class
0    0.998274
1    0.001726
Name: proportion, dtype: float64
Validation Set Class Distribution:
Class
0    0.998262
1    0.001738
Name: proportion, dtype: float64
Test Set Class Distribution:
Class
0    0.99828
1    0.00172
Name: proportion, dtype: float64
```

Figure 1: Data Split

Figure 3: Class Distribution

Special Characteristics

- An important challenge was dealing with the class imbalance and tuning the hyperparameters.
- Through our analysis, we found there were no missing data entries, so no imputation was required.

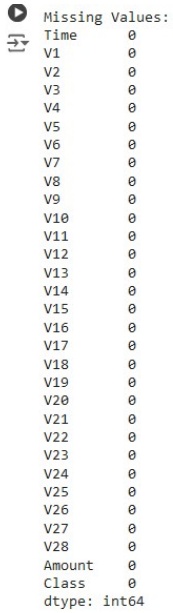


Figure 2: Missing Values

- The data is anonymized, meaning the PCA-transformed features are not directly understandable, so key user data was not accessible.
- While the anonymized features were on the same scale, we needed to scale the *Amount* and *Time* features because their original values were in different ranges compared to the other features.

We then split the dataset into a training set and a test set. 80% of the dataset was split into the training-validation set for training and hyperparameter tuning. The test set was 20% of the dataset for final evaluation. We further split the training set into 75% for training and 25% for validation.

The next step in our process was to implement Synthetic Minority Oversampling Technique (SMOTE). SMOTE is an oversampling technique used to address class imbalance in datasets, particularly in machine

learning. It generates synthetic examples for the minority class to balance the class distribution. It selects samples from the minority class, then for each selected minority class instance, the algorithm identifies its k-nearest neighbors within the same class using a distance metric like Euclidean distance. A new synthetic sample is created by interpolating between the selected minority instance and one of its nearest neighbors. This process is repeated until the desired balance between the majority and minority classes is achieved.

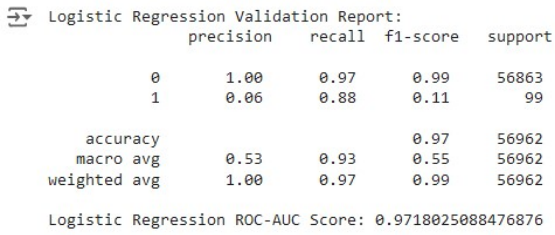
We decided to use this step to test whether it improved the model training by balancing the dataset to try and prevent the model from being biased toward the majority class.

Algorithm Description

We started with three algorithms - Logistic Regression, Random Forest, and XGBoost.

- Logistic Regression is a simple and interpretable model that acts as a baseline for comparison.
- Random Forest is a robust model that handles non-linear patterns and provides feature importance insights.
- XGBoost is a high-performing gradient boosting algorithm that handles class imbalance well with the `scale_pos_weight` parameter.

We tested the models on the SMOTE data first and printed out the results:



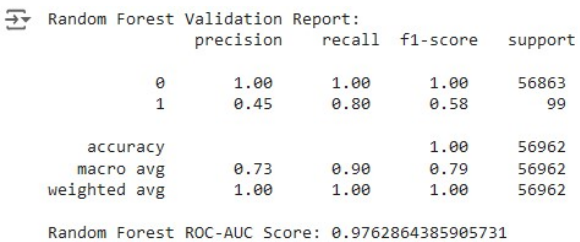
```
Logistic Regression Validation Report:
              precision    recall  f1-score   support

           0       1.00      0.97      0.99     56863
           1       0.06      0.88      0.11        99

    accuracy                           0.97     56962
   macro avg       0.53      0.93      0.55     56962
weighted avg       1.00      0.97      0.99     56962

Logistic Regression ROC-AUC Score: 0.9718025088476876
```

Figure 4: Linear Regression with SMOTE



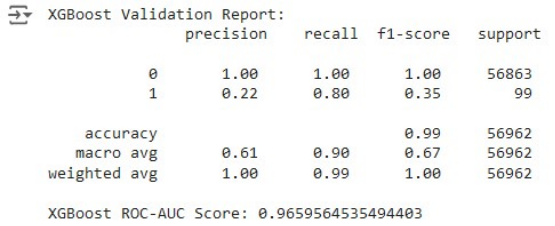
```
Random Forest Validation Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56863
           1       0.45      0.80      0.58        99

    accuracy                           1.00     56962
   macro avg       0.73      0.90      0.79     56962
weighted avg       1.00      1.00      1.00     56962

Random Forest ROC-AUC Score: 0.9762864385905731
```

Figure 5: Random Forest with SMOTE



```
XGBoost Validation Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56863
           1       0.22      0.80      0.35        99

    accuracy                           0.99     56962
   macro avg       0.61      0.90      0.67     56962
weighted avg       1.00      0.99      1.00     56962

XGBoost ROC-AUC Score: 0.9659564535494403
```

Figure 6: XGBoost with SMOTE

Then we tested the data without the SMOTE technique and compared the result:

```
Logistic Regression Validation Report (No SMOTE):
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     56863
           1       0.06      0.90      0.11        99

    accuracy                           0.97     56962
   macro avg       0.53      0.94      0.55     56962
weighted avg       1.00      0.97      0.99     56962

Logistic Regression ROC-AUC Score (No SMOTE): 0.9746129142221505
```

Figure 7: Linear Regression without SMOTE

```
Random Forest Validation Report (No SMOTE):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56863
           1       0.87      0.76      0.81        99

    accuracy                           1.00     56962
   macro avg       0.94      0.88      0.91     56962
weighted avg       1.00      1.00      1.00     56962

Random Forest ROC-AUC Score (No SMOTE): 0.9707846450719672
```

Figure 8: Random Forest without SMOTE

```
XGBoost Validation Report (No SMOTE):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56863
           1       0.83      0.78      0.80        99

    accuracy                           1.00     56962
   macro avg       0.91      0.89      0.90     56962
weighted avg       1.00      1.00      1.00     56962

XGBoost ROC-AUC Score (No SMOTE): 0.9696522050073569
```

Figure 9: XGBoost without SMOTE

The models were evaluated based on their F1-score, precision, and recall on their validation sets. We also made sure to use accuracy, macro average, weighted average, and computed the ROC-AUC score for each algorithm to make an informed decision on the final algorithm to use on the test set.

Model Evaluation

The results showed:

- **Logistic Regression:** High recall but low precision, resulting in a poor F1-score for fraud detection. We used 2 hyper parameters to tune this model:
  - **Class Weight** (class_weight='balanced') This hyper parameter means that the model accounted for the extreme class imbalance by assigning higher importance to fraudulent transactions during training. If we did not add this the model would heavily favor the majority class.
  - **Regularization Strength** (C) We chose a smaller C value to help improve generalization. It controls the trade-off between achieving a low error on the training data and regularizing the model to avoid overfitting.

  - While the Logistic Regression model showed interpretable results, it struggled with this dataset because of the algorithm's linear nature. Tuning improved recall (0.90 for fraud) but precision was low (0.06) which resulted in a low F1-Score (0.11). However, this did assist us as a baseline for comparison.

- **Random Forest:** Best-performing model, with an F1-score of 0.81 for fraudulent transactions and a balanced trade-off between precision and recall. We used 4 hyper parameters to tune this model:
  - **Number of trees** (n_estimators) made sure that the model had enough trees to capture the patterns without being too computationally expensive.
  - **Class Weight** (class_weight='balanced') adjusted the weights to prioritize the minority class.
  - **Maximum Depth** (max_depth) control tree depth to prevent overfitting.
  - **Minimum Samples per Leaf** (min_samples_leaf) helped to make sure that each leaf node represented a sufficient number of samples and made sure there weren't splits on noisy data.
  - Random Forest was able to handle the non-linear patterns in the dataset and the class imbalance.
- **XGBoost:** Comparable performance to Random Forest but slightly lower recall and F1-score. We used 5 hyper parameters:
  - **Learning Rate** (eta) this determines the step size for each iteration with a smaller value chosen to reduce overfitting and improve convergence.
  - **Number of Boosting Rounds** (n_estimators) controls the number of iterations, balancing the model complexity and runtime.
  - **Maximum Depth** (max_depth) Limited the depth of the trees so the data was not overfit.
  - **Scale Positive Weight** (scale_pos_weight) handles the class imbalance by increasing the weight of the minority class.
  - **Gamma** controlled min loss reduction that was required to split nodes which helped avoid splits on features that were insignificant.
  - XGBoost ended up being comparable to the Random Forest model, but had slightly lower recall and F1-Score. It handled the class-imbalance well but required more careful tuning due to it being sensitive to overfitting.

Random Forest emerged as the best-performing algorithm for this dataset, both with and without SMOTE, due to its ability to handle class imbalance and non-linear patterns effectively.

Performance Metrics

Precision: Random Forest had the highest precision, meaning it made fewer false positive predictions (fewer legitimate transactions flagged as fraud).
F1-Score: Its F1-Score indicates the best balance between precision and recall among the models.
Recall: It detected a large percentage of fraud cases while maintaining a strong precision.

Model Robustness

Random Forest handles imbalanced datasets well by using `class_weight='balanced'`.
It is less sensitive to noise or irrelevant features compared to XGBoost, which might have been affected by the nature of this dataset.

Ease of Use

Random Forest performed well with minimal hyperparameter tuning, making it easier to optimize.

Comparative Results

Random Forest outperformed Logistic Regression and XGBoost in most metrics, especially in precision and F1-Score, which are critical for fraud detection.

As a result, we used the Random Forest classifier to predict on the test data. Out of 98 fraudulent transactions in the test set, the model correctly identified 81 cases as fraud while only misclassifying 17 cases as legitimate. With just 16 false alarms (false positives) out of over 56,800 legitimate transactions, the model provides reliable predictions with minimal disruption to legitimate users.

```
Random Forest Test Set Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.84      0.83      0.83        98

    accuracy                           1.00     56962
   macro avg       0.92      0.91      0.92     56962
weighted avg       1.00      1.00      1.00     56962

Random Forest Test Set ROC-AUC Score: 0.9712937527993752
```

Figure 10: Random Forest on Test Data

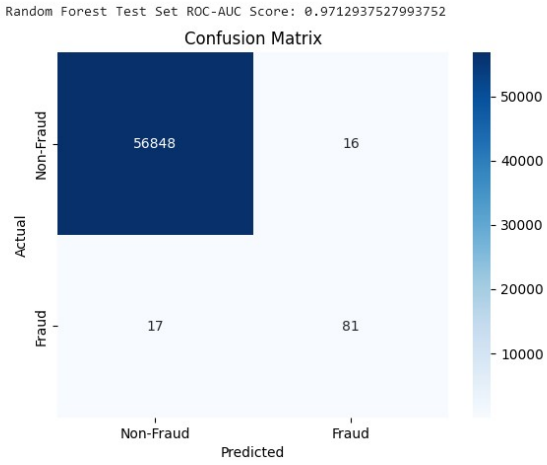


Figure 11: Confusion Matrix

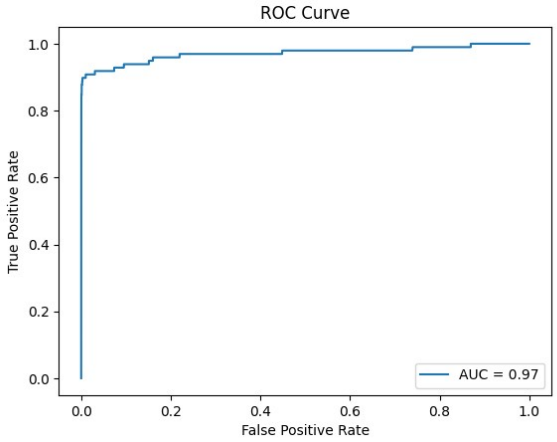


Figure 12: ROC Curve

The model had high precision (0.84) and recall (0.83) for fraud detection, resulting in the best F1-Score (0.83). Furthermore, it had a high ROC-AUC score (0.971) which shows a strong discriminatory power between fraud and non-fraud transactions. Random forest also resulted in minimal false positive (16) and false negatives (17) making it suitable for real-world applications. Fraud detection is a tough task because the datasets are often imbalanced, but Random Forest handled it well in this scenario when using balanced class weights. By assigning weights inversely proportional to the class frequencies, this approach helped to ensure that the minority class, in this case the fraudulent transactions, received greater importance during the model training.

Conclusion

This project highlighted the importance of addressing class imbalance in fraud detection. The following conclusions were drawn:

- Random Forest proved to be the most effective model, achieving a balance between precision and recall. The model is scalable and can be deployed in a real world scenario.
- SMOTE improved model performance by generating synthetic samples for the minority class.
- Comparing multiple algorithms and systematically tuning hyper parameters is critical for optimizing performance.

The Random Forest model performed exceptionally well in detecting fraudulent transactions while maintaining high accuracy for legitimate transactions. Precision (84%): Demonstrates the model's ability to avoid false alarms (false positives). Recall (83%): Indicates the model successfully detected the majority of fraud cases, though some cases were missed (17 false negatives). ROC-AUC Score (0.971): Highlights the model's excellent ability to distinguish between fraudulent and legitimate transactions. Fraud Detection Success: Out of 98 fraudulent transactions in the test set, the model correctly identified 81 cases as fraud while only misclassifying 17 cases as legitimate. With just 16 false alarms (false positives) out of over 56,800 legitimate transactions, the model provides reliable predictions with minimal disruption to legitimate users. Balance Between Precision and Recall: The model achieved a good balance between Precision and Recall, ensuring both accurate fraud detection and minimal false alarms, making it well-suited for real-world deployment in fraud detection systems.

Future work could explore advanced oversampling techniques and hybrid models to further enhance fraud detection. Also, experimenting with threshold tuning to reduce False Negatives (missed fraud cases) to try and improve recall. Another idea could be to deploy Random Forest model using Flask/FastAPI or AWS SageMaker for real-time fraud detection.